UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/716,099	11/17/2003	Byron D. Vargas	03917-P0001B	6113

24126 7590 05/09/2007

ST. ONGE STEWARD JOHNSTON & REENS, LLC
986 BEDFORD STREET
STAMFORD, CT 06905-5619

EXAMINER	
NGUYEN, PHILLIP H	
ART UNIT	**PAPER NUMBER**
2191	
MAIL DATE	**DELIVERY MODE**
05/09/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/716,099	VARGAS, BYRON D.	
	Examiner	Art Unit	
	Phillip H. Nguyen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *01 March 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-48* is/are pending in the application.

4a) Of the above claim(s) *1,3,16-24,34-42 and 46* is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *2,4-9,11-15,25-33,43-45,47 and 48* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *20031117*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is in response to the amendment filed on March 01, 2007.

2. Per Applicant's request, claims 2, 4-9, 11-15, 25, 43, 44, 47 and 48 have been amended. Claims 1, 3, 16-24, 34-42 and 46 have been cancelled.

3. Claims 2, 4-15, 25-33, 43-45, 47 and 48 remain pending and have been considered below.

Claim Rejections - 35 USC § 101

4. The amendment filed on March 01, 2007 overcomes the 101 rejection to claims 14 and 43 of the previous action. Therefore, the 101 rejection for software per se is withdrawn.

Claim Rejections - 35 USC § 112

5. The amendment filed on March 01, 2007 overcomes the 112 rejection to claims 3, 6, 16, 35 and 44 of previous action. Therefore, the rejection is withdrawn.

Response to Arguments

6. Applicant's arguments with respect to claims 2, 4-9, 11-15, 25, 43, 44, 47 and 48 have been considered but are moot in view of the new ground(s) of rejection.

Claim Rejections - 35 USC § 102

7. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

8. Claims 14, 2, 4, 7-10 and 15 are rejected under 35 U.S.C. 102(b) as being anticipated by Laitila (International Application Published Under The PCT No.: WO 02/093371 A1).

As per claim 14:

Laitila discloses:

- a computer having a storage (see at least page 3, line 26 "**a computer system**");
- an emulated Application Programming Interface library having a table and accessible by said computer translating system, said library including data indicative of types of data manipulations between the first computer language source code and the second computer language source code, said table including second computer language equivalent functions (see at least page 6, line 16 "**an operations library, in which are the routines to be called from the translator (X>Y)**");
- an analyzer analyzing the type of data manipulation that the first computer language source code performs, said analyzer accessing said table of said emulation library (see at least page 7, line 28-25 "**initially, the occurrence of**

"add" corresponding to addition is selected from the input language. The developer may propose "math_oper" as the corresponding occurrence in the target language, as the term "+" appears in the syntax of both") and correlating the type of data manipulation the first computer language source code performs to a second computer language equivalent function (see at least Figure 4; also see at least page 7, line 21-24 "....**the first parameter is the data structure of input language and the second parameter is the data structure of the target language...the intention is to get them to correspond to each other i.e., to create a semantic connection between them**");

- a generator for generating second computer language source code based on the identified equivalent functions such that second computer language source code that emulates the type of data manipulation the first computer language source code performs (see at least page 7, line 36-40 and page 8, line 1-4 "**finally, the button "develop code" is clicked, when the developer will create the necessary conversion instruction by generating the lower-level conversion calls for the conversion between input and target terms. Final result obtained is a PROGLOG-language predicate**..."); and
- said second computer language source code providing discreet functionality that is independent from the first computer language source code such that said generated second computer language source code can independently provide the equivalent type of data manipulations provided in said first computer language source code without reference to the first computer language source

code (see at least page 1, 1st paragraph "...**two source languages independent of each other**...").

As per claim 2:

Laitila discloses:

- wherein the translator is a bi-directional translator, said analyzer analyzing the second computer language source code to identify the type of data manipulation that the second computer language source code performs, accessing said emulation library and correlating the type of data manipulation the second computer language source code performs to first computer language source code, the correlation being independent of the context in which the second computer language source code is used, said software generating re-translated first computer language source code that emulates the type of data manipulation the second computer language source code performs (see at least page 2, line 39 and page 3, line 1-2 "**the method according to the invention provides an opportunity to test the method by converting the program code translated to the intermediate language back to the input language**").

As per claim 4:

Laitila discloses:

- wherein said generator performs a name adjustment when an incompatibility between the first computer language source code and the second computer

language source code occurs during correlation of the type of data manipulation the first computer language source code performs to the second computer language source code to resolve the incompatibility, the name adjustment being independent of the context in which the incompatibility is located (see at least page 7, line 24-26 "**compatibility is obtained by clicking the Y-language parameters so many times that the situation is reached in which the number of the parameters of the X and Y languages are the same**").

As per claim 7:

Laitila discloses:

- wherein the first computer language source code comprises an identifier (see at least page 1, line 27 "**keywords**").

As per claim 8:

Laitila discloses:

- wherein said software performs a name adjustment when an incompatibility between the identifier and the second computer language source code occurs during correlation of the type of data manipulation the identifier performs to the second computer language source code to resolve the incompatibility, the name adjustment being independent of the context in which the identifier is used (see at least page 7, line 24-26 "**compatibility is obtained by clicking the Y-**

language parameters so many times that the situation is reached in which the number of the parameters of the X and Y languages are the same").

As per claim 9:

Laitila discloses:

- wherein said software generates a tagged element inserted in the second computer language source code indicative of a type of data manipulation the first computer language source code performs (see page 10-41).

As per claim 10:

Laitila discloses:

- wherein the tagged element comprises information selected from the group consisting of: formatting, translation data, and first computer language source code (see page 10-41).

As per claim 15:

Laitila discloses:

- wherein the translating system is a bi-directional translator ((see at least page 2, line 39 and page 3, line 1-2 "**the method according to the invention provides an opportunity to test the method by converting the program code translated to the intermediate language back to the input language**"), and further comprises a parser parsing the first and second computer language

source codes into parsed elements (see at least page 1, line 24-26 "**a parser operation is used to group the key terms to form a parser tree, which is then parsed to give individual terms, for the processing of which specific rules are created**").

Claim Rejections - 35 USC § 103

9. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

10. Claims 5, 6, 11-13, 25-33, 43-45, 47 and 48 are rejected under 35 U.S.C. 103(a) as being unpatentable over Laitila (International Application Published Under The PCT No.: WO 02/093371 A1).

As per claim 5:

Laitila does not explicitly disclose:

- wherein the first computer language source code comprises a class.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitila's approach is applicable to object oriented programming. One would have been motivated to use Laitila's translator to convert Object Pascal language (the extended Pascal was designed for object oriented

programming) to C or C++ because the first language in Laitila's approach is Pascal programming language (see at least page 8, line 38).

As per claim 6:

Laitila does not explicitly disclose:

- wherein the class consists of: methods, data fields, inner-classes, and combination thereof.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that the Object Pascal, the newer version of Pascal designed for object oriented programming includes class that consists of methods, data fields, inner-classes and combination thereof.

As per claim 11:

Laitila does not explicitly disclose:

- wherein the first computer language is Java and the second computer language is C++.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitia's approach is applicable to any high level programming languages since Laitila's approach is not limited to just Pascal and C languages. One would have been motivated to use Laitila's translator to convert Java to C++ because Pascal, C, Java and C++ are all similar in syntax rules.

As per claim 12:

Laitila does not explicitly disclose:

- wherein the first computer language is Java and the second computer language is C#.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitia's approach is applicable to any high level programming languages since Laitila's approach is not limited to just Pascal and C languages. One would have been motivated to use Laitila's translator to convert Java to C# because Pascal, C, Java and C# are all similar in syntax rules.

As per claim 13:

Laitila does not explicitly disclose:

- wherein the first computer language is C# and the second computer language is C++.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitia's approach is applicable to any high level programming languages since Laitila's approach is not limited to just Pascal and C languages. One would have been motivated to use Laitila's translator to convert C# to C++ because Pascal, C, Java, C# and C++ are all similar in syntax rules.

As per claim 25:

Laitila discloses:

- inputting the first computer language source code into a computer (see at least page 1, line 23 "**the code is read using a scanner operation**");
- parsing the first computer language source code into parsed elements (see at least page 1, line 23-25 "**a parser operation is used to group the key terms to form a parser tree, which is then parsed to give individual terms, for processing of which specific rules are created**");
- analyzing the parsed elements to determine a type of data manipulation the first computer language source code performs (see at least page 7, line 28-25 "**initially, the occurrence of "add" corresponding to addition is selected from the input language. The developer may propose "math_oper" as the corresponding occurrence in the target language, as the term "+" appears in the syntax of both**");
- referencing an emulation Application Programming Interface library including second computer language equivalent functions that emulate the first computer language data manipulations (see at least page 6, line 16 "**an operations library, in which are the routines to be called from the translator (X>Y)**");
- selecting second computer language equivalent functions from the emulation library that emulate the first computer language data manipulations (see at least Figure 4; also see at least page 7, line 21-24 "....**the first parameter is the data structure of input language and the second parameter is the data structure**

of the target language...the intention is to get them to correspond to each other i.e., to create a semantic connection between them").

Laitila does not explicitly disclose:

- building class declarations and class definitions from the parsed elements that independently provide the equivalent type of data manipulations provided in the first computer language source code without reference to the first computer language source code;
- generating the second computer language source code according to the class declarations and class definitions, the second computer language source code emulating the type of data manipulation the first computer language source code performs.

However, it would have obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitia's approach is applicable to any high level programming languages since Laitila's approach is not limited to just Pascal and C languages. One would have been motivated to use Laitila's translator to convert Object Pascal to C, Java, C# or C++ because Pascal, Object Pascal, C, Java, C# and C++ are similar in syntax rules. Therefore, building class declaration and definitions from the parsed elements and generating the second computer language source code according to the class declaration and definitions would be part of Laitila's approach.

As per claim 26:

Laitila discloses:

- parsing the second computer language source code into parsed elements; analyzing the parsed elements to determine a type of data manipulation the second computer language source code performs; building class declarations and class definitions from the parsed elements that are independent of the context of the second computer language source code; generating the re-translated first computer language source code according to the class declarations and class definitions, the first computer language source code emulating the type of data manipulation the second computer language source code performs (see at least page 2, line 39 and page 3, line 1-2 "**the method according to the invention provides an opportunity to test the method by converting the program code translated to the intermediate language back to the input language**").

As per claim 27:

Laitila discloses:

- performing a name adjustment when an incompatibility between the first computer language source code and the second computer language source code occurs during translation, the name adjustment being independent of the context in which the incompatibility is located (see at least page 7, line 24-26 "**compatibility is obtained by clicking the Y-language parameters so many**

times that the situation is reached in which the number of the parameters of the X and Y languages are the same").

As per claim 28:

Laitila discloses:

- wherein the first computer language source code comprises an identifier (see at least page 1, line 27 "**keywords**").

As per claim 29:

Laitila discloses:

- performing a name adjustment when an incompatibility between the identifier and the second computer language source code occurs during translation, the name adjustment being independent of the context in which the identifier is located (see at least page 7, line 24-26 "**compatibility is obtained by clicking the Y-language parameters so many times that the situation is reached in which the number of the parameters of the X and Y languages are the same**").

As per claim 30:

Laitila discloses:

- generating a tagged element indicative of the type of data manipulation the first computer language source code performs; and inserting the tagged element in the second computer language source code (see page 10-41).

As per claim 31:

Laitila does not explicitly disclose:

- wherein the first computer language is Java and the second computer language is C++.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitia's approach is applicable to any high level programming languages since Laitila's approach is not limited to just Pascal and C languages. One would have been motivated to use Laitila's translator to convert Java to C++ because Pascal, C, Java and C++ are all similar in syntax rules.

As per claim 32:

Laitila does not explicitly disclose:

- wherein the first computer language is Java and the second computer language is C#.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitia's approach is applicable to any high level programming languages since Laitila's approach is not limited to just Pascal and C languages. One would have been motivated to use Laitila's translator to convert Java to C# because Pascal, C, Java and C# are all similar in syntax rules.

As per claim 33:

Laitila does not explicitly disclose:

- wherein the first computer language is C# and the second computer language is C++.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitia's approach is applicable to any high level programming languages since Laitila's approach is not limited to just Pascal and C languages. One would have been motivated to use Laitila's translator to convert C# to C++ because Pascal, C, Java, C# and C++ are all similar in syntax rules.

As per claim 43:

Laitila discloses:

- a computer having a storage (see at least page 3, line 26 "a computer system");
- an emulation Application Interface library including data indicative of types of data manipulations between the Java source code and the OOP language source code, said table including OOP language equivalent functions (see at least page 6, line 16 "**an operations library, in which are the routines to be called from the translator (X>Y)**");
- an analyzer analyzing the Java source code to determine a type of data manipulation the Java source code performs, said analyzer accessing said table of said emulation library (see at least page 7, line 28-25 "**initially, the occurrence of "add" corresponding to addition is selected from the input**

language. The developer may propose "math_oper" as the corresponding occurrence in the target language, as the term "+" appears in the syntax of both") and correlating the type of data manipulation the Java source code performs to a OOP language equivalent function (see at least Figure 4; also see at least page 7, line 21-24 "....**the first parameter is the data structure of input language and the second parameter is the data structure of the target language...the intention is to get them to correspond to each other i.e., to create a semantic connection between them**");

- a generator for generating OOP language source code based on the identified equivalent functions such that OOP language source code emulates the type of data manipulation the Java source code performs (see at least page 7, line 36-40 and page 8, line 1-4 "**finally, the button "develop code" is clicked, when the developer will create the necessary conversion instruction by generating the lower-level conversion calls for the conversion between input and target terms. Final result obtained is a PROGLOG-language predicate**..."); and
- said OOP language source code providing discreet functionality that is independent from the Java source code such that said generated OOP source code can independently provide the equivalent type of data manipulation provided in said Java source code without reference to Java source code (see at least page 1, 1st paragraph "...**two source languages independent of each other**...").

Laitila does not explicitly disclose translating Java source code to OOP.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitia's approach is applicable to any high level programming languages since Laitila's approach is not limited to just Pascal and C languages. One would have been motivated to use Laitila's translator to convert Java to C because Pascal, C, Java, C# and C++ are similar in syntax rules.

As per claim 44:

Laitila discloses:

- wherein said analyzer analyzes the OOP language source code to determine a type of data manipulation the OOP language source code performs, said analyzer accessing said library to correlate the type of data manipulation the OOP language source code performs to Java source code, the correlation being independent of the context in which the OOP language source code is used, and said generator generates re-translated Java source code that emulates the type of data manipulation the OOP language source code performs, the re-translated Java source code being substantially identical to the Java source code (see at least page 2, line 39 and page 3, line 1-2 "**the method according to the invention provides an opportunity to test the method by converting the program code translated to the intermediate language back to the input language**").

As per claim 45:

Laitila does not explicitly disclose:

- wherein said analyzer builds class declarations and class definitions based on the type of data manipulation the Java source code performs and said generator generates OOP language source code based upon the class declarations and class definitions.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitia's approach is applicable to any high level programming languages since Laitila's approach is not limited to just Pascal and C languages. One would have been motivated to use Laitila's translator to convert Java to C because Pascal, C, Java, C# and C++ are similar in syntax rules. Therefore, the above limitation would be part of Laitila's approach.

As per claim 47:

Laitila discloses:

- analyzing the first OOP computer language source code to determine a type of data manipulation the first OOP computer language source code performs (see at least page 7, line 28-25 "**initially, the occurrence of "add" corresponding to addition is selected from the input language. The developer may propose "math_oper" as the corresponding occurrence in the target language, as the term "+" appears in the syntax of both**");

- referencing an emulation Application Programming Interface library including second OOP computer language equivalent functions that emulate the first OOP computer language data manipulations (see at least page 6, line 16 "**an operations library, in which are the routines to be called from the translator (X>Y)**");
- correlating the type of data manipulation of the first OOP computer language source code performs to a second OOP computer language equivalent function (see at least Figure 4; also see at least page 7, line 21-24 "....**the first parameter is the data structure of input language and the second parameter is the data structure of the target language...the intention is to get them to correspond to each other i.e., to create a semantic connection between them**");
- generating the second OOP computer language source code based on the identified equivalent functions such that the second OOP language source code emulates the type of data manipulation the first OOP computer language source code performs (see at least page 7, line 36-40 and page 8, line 1-4 "**finally, the button "develop code" is clicked, when the developer will create the necessary conversion instruction by generating the lower-level conversion calls for the conversion between input and target terms. Final result obtained is a PROGLOG-language predicate**..."); and
- performing a data manipulation with the second OOP computer language source code that emulates the type of data manipulation the first OOP computer

language source code performs, the second OOP computer language source code performing the data manipulation without use of a virtual machine (see at least page 7, line 28-25 "**initially, the occurrence of "add" corresponding to addition is selected from the input language. The developer may propose "math_oper" as the corresponding occurrence in the target language, as the term "+" appears in the syntax of both**");

Laitila does not explicitly disclose the first language is OOP language.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitia's approach is applicable to any high level programming languages since Laitila's approach is not limited to just Pascal and C languages. One would have been motivated to use Laitila's translator to convert Object Pascal to C because Object Pascal is an extended version of Pascal for supporting object oriented programming.

As per claim 48:

Laitila discloses:

- analyzing the first OOP computer language source code to determine a type of data manipulation the first OOP computer language source code performs (see at least page 7, line 28-25 "**initially, the occurrence of "add" corresponding to addition is selected from the input language. The developer may propose "math_oper" as the corresponding occurrence in the target language, as the term "+" appears in the syntax of both**");

- referencing an emulation Application Programming Interface library including second OOP computer language equivalent functions that emulate the first OOP computer language data manipulations (see at least page 6, line 16 "**an operations library, in which are the routines to be called from the translator (X>Y)**");
- correlating the type of data manipulation of the first OOP computer language source code performs to second OOP computer language source code (see at least Figure 4; also see at least page 7, line 21-24 "....**the first parameter is the data structure of input language and the second parameter is the data structure of the target language...the intention is to get them to correspond to each other i.e., to create a semantic connection between them**");
- generating the second computer language source code based on the identified equivalent functions such that the second OOP language source code emulates the type of data manipulation the first OOP computer language source code performs (see at least page 7, line 36-40 and page 8, line 1-4 "**finally, the button "develop code" is clicked, when the developer will create the necessary conversion instruction by generating the lower-level conversion calls for the conversion between input and target terms. Final result obtained is a PROGLOG-language predicate**..."); and
- performing a data manipulation with the second computer language source code that emulates the type of data manipulation the first computer language source code performs, the second computer language source code performing the data

manipulation without use of a garbage collector (see at least page 7, line 28-25 **"initially, the occurrence of "add" corresponding to addition is selected from the input language. The developer may propose "math_oper" as the corresponding occurrence in the target language, as the term "+" appears in the syntax of both**");

Laitila does not explicitly disclose the first language is OOP language.

However, it would have been obvious to one having an ordinary skill in the art at the time the invention was made to recognize that Laitia's approach is applicable to any high level programming languages since Laitila's approach is not limited to just Pascal and C languages. One would have been motivated to use Laitila's translator to convert Object Pascal to C because Object Pascal is an extended version of Pascal for supporting object oriented programming.

Conclusion

11. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of

the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Phillip H. Nguyen whose telephone number is (571) 270-1070. The examiner can normally be reached on Monday - Thursday 10:00 AM - 3:00 PM EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Wei Y. Zhen can be reached on (571) 272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

WEI ZHEN
SUPERVISORY PATENT EXAMINER

PN
04/23/2007